Graphical Models 84 (2016) 28–37

Contents lists available at ScienceDirect

Graphical Models

journal homepage: www.elsevier.com/locate/gmod

Fast exact shortest distance queries for massive point clouds

David Eriksson[a,1,*], Evan Shellshear[b]

[a] Center for Applied Mathematics, Cornell University, Ithaca, New York, USA
[b] Fraunhofer-Chalmers Centre, Gothenburg, Sweden

ARTICLE INFO

Article history:
Received 2 September 2015
Revised 24 February 2016
Accepted 26 February 2016
Available online 19 March 2016

Keywords:
Massive point cloud
Shortest distance computation
Out-of-core
Path-planning

ABSTRACT

This paper describes a new efficient algorithm for the rapid computation of exact shortest distances between a point cloud and another object (e.g. triangulated, point-based, etc.) in three dimensions. It extends the work presented in Eriksson and Shellshear (2014) where only approximate distances were computed on a simplification of a massive point cloud. Here, the fast computation of the exact shortest distance is achieved by pruning large subsets of the point cloud known not to be closest to the other object. The approach works for massive point clouds even with a small amount of RAM and is able to provide real time performance. Given a standard PC with only 8GB of RAM, this resulted in real-time shortest distance computations of 15 frames per second for a point cloud having 1 billion points in three dimensions.

1. Introduction

High-resolution point clouds have become very important in the last decades as researchers have started to exploit their advantages over triangle-based models in computer graphics applications [19,21]. Improvements and significant price decreases in laser scanning technologies have made it possible to easily and cheaply scan very large objects, thereby creating massive quantities of point cloud data. At the same time, point clouds now offer significant advantages over traditional CAD geometries for real life applications and simulations, Eriksson and Shellshear [6]. In particular, one area which has received more attention is path planning in large point clouds and the ability to quickly compute proximity queries (collision, distance, etc.).

Path-planning through environments consisting of triangle meshes has been studied intensively and there exists a significant amount research on the subject [3,7,9,10,16,17]. Although the area of path-planning with point clouds is newer, there are methods designed specifically for it, such as [11].

Our focus in this paper is on the distance computation aspects of path planning. In this area, there are no known papers to the authors for computing the distances between dynamic massive triangle meshes and point clouds which do not fit in main memory. The Octomap data structures were used in [15] to compute distances between point clouds, but their focus was not on massive data sets so the methods used there were not directly applicable here. Distance computations between massive static point clouds and other objects can be carried out with out-of-core exact nearest neighbor queries, [2]. For a more thorough review of the literature we refer to [6].

In this paper we extend the important work begun in [6] where preprocessing algorithms were developed to divide a point cloud into smaller subclouds and also simplify the point cloud to allow approximate shortest distance computations to be carried out in main memory. These methods demonstrated how fast and approximate closest distance computations could be carried out for massive point clouds which do not fit into main memory.

However, a drawback of the work presented in [6] is the tradeoff between the size of the point cloud and the level of simplification necessary. Already on a point cloud with one billion points it was shown to be necessary to accept an error of up to 2 cm to fit the point cloud and other necessary data structures into main memory. As the point clouds become larger, this error only increases. Here, we show a better way to compute distances between massive point clouds and other objects (point clouds, CAD geometries, etc.), which is scalable to point clouds of any size and also has no error in the distance computation, i.e., all distance computations are exact.

This article demonstrates how it is possible to perform fast, exact distance queries for path-planning applications in massive point clouds independent of the amount of RAM available. Because our final usage is in real-time applications, like in [6], our goal is to achieve shortest distance computations with a frame rate of around 15 fps.

* Corresponding author.
E-mail address: dme65@cornell.edu (D. Eriksson).
[1] This research was carried out while at Fraunhofer Chalmers Centre, Sweden.

http://dx.doi.org/10.1016/j.gmod.2016.02.002
1524-0703/© 2016 Elsevier Inc. All rights reserved.

In order to do this we start by dividing the point cloud into disjoint subsets using the methods presented in [6]. Given these subsets, we construct the convex hull of each subset and build two Proximity Query Package (PQP) models [12] for each subset, where one contains the extreme points and the other the non-extreme points. We then introduce Theorems 2 and 4 in order to decide which subsets can contain the point closest to the object and only these PQP models will have to reside in RAM. Our focus here is on three dimensions but, in theory, we expect the results to be applicable also in higher dimensions. In practice the generalization to higher dimensions will be challenging due to the computational and geometric complexity of the convex hull in higher dimensions. Our proposed strategy is shown to achieve our goal of a fast frame rate with massive point clouds. One limitation of the algorithm is that it does not allow the object moving through the point cloud to be initially contained in the convex hull of one of the subsets.

Section 2 describes how to pre-process a point cloud for fast exact distance queries. Section 3 introduces a simple way of using information from the previous distance computation to exclude subsets that cannot contain the point closest to the object. The main algorithms are presented in Section 4 and in Section 5 we test the theoretical results from the first two sections on a real-life scenario.

2. Pre-processing of the point cloud for fast distance queries

This section will describe how the convex hull of a subset of points can be used to derive a lower bound on how close any points contained in the convex hull can be to the object. If we know that there is a point that is closer to the object than a given lower bound for the distance to a given convex hull, then this subset of points can be excluded and therefore computing the shortest distance from this subset to the object can be avoided. This lower bound is based on the extreme points of the convex hulls and allows for a fast way to approximate the distance to the object from a set of points.

2.1. Fast distance approximation

Let Q be a finite point cloud, i.e., $Q = \{q^1, \ldots, q^m \mid q^i \in \mathbb{R}^n, i = 1, \ldots, m\}$ and recall the definition of the convex hull containing the interior:

Definition 1. The convex hull of a set of points Q is the set

$$\mathcal{CH}(Q) = \left\{ \sum_{i=1}^{|Q|} \lambda_i q^i \mid \lambda_i \ge 0, \sum_{i=1}^{|Q|} \lambda_i = 1 \right\}. \tag{1}$$

We denote the boundary of $\mathcal{CH}(Q)$ by $\partial\mathcal{CH}(Q)$ and assume throughout the article that $\mathcal{CH}(Q)$ is full-dimensional, i.e., $\dim(\mathcal{CH}(Q)) = \dim(Q) = n$, although the arguments presented here can be adapted to the case that $\mathcal{CH}(Q)$ is not full-dimensional. Denote by $S \subset \mathbb{R}^n$ the object which is moved through the point cloud and assume that this object is a compact set (e.g. it could be a polygon mesh or a point cloud). Other than this, we assume nothing else of the set S. We now introduce the following convenient notation for the distance between two points, from a point to a set, and from a set to a set:

Definition 2. Let $Q, R \subset \mathbb{R}^n$ and $q, r \in \mathbb{R}^n$. Define:

$$\begin{aligned} d(q,r) &= \|q - r\|_2 \\ d(q,R) &= \inf_{r \in R} d(q,r) \\ d(Q,R) &= \inf_{\substack{q \in Q \\ r \in R}} d(q,r). \end{aligned} \tag{2}$$

It is worth mentioning that the triangle inequality does not hold for general Q and R, e.g. when both are arbitrary sets, but it remains valid here as long as points are introduced as intermediates. As a preparation for the main results, an intuitively obvious theorem stating that the point belonging to $\mathcal{CH}(Q)$ that is closest to S must lie in $\partial\mathcal{CH}(Q)$ is proved next. Although this result seems to be a consequence of other well-known results we present a proof here for completeness.

Theorem 1. *Let Q be a finite point cloud and $S \subset \mathbb{R}^n$ be closed and bounded and assume that $\mathcal{CH}(Q)$ is full-dimensional. If $\mathcal{CH}(Q) \cap S = \emptyset$, the point in $\mathcal{CH}(Q)$ closest to S must lie on the boundary.*

Proof. Let $V = (S, \mathcal{CH}(Q)) \subseteq \mathbb{R}^n \times \mathbb{R}^n \cong \mathbb{R}^{2n}$ and define the function

$$\begin{aligned} &f : \mathbb{R}^{2n} \to \mathbb{R} \\ &f(x) = \left\| x_S - x_{\mathcal{CH}(Q)} \right\|_2 \end{aligned} \tag{3}$$

where $x_S \in S$ and $x_{\mathcal{CH}(Q)} \in \mathcal{CH}(Q)$. From Weierstrass' theorem, [14],

$$\exists y = (y_S, y_{\mathcal{CH}(Q)}) \subseteq \mathbb{R}^n \times \mathbb{R}^n \cong \mathbb{R}^{2n}, \text{ such that } y = \arg\min_{x \in V} f(x). \tag{4}$$

since V is closed and bounded. If $y_{\mathcal{CH}(Q)}$ lies on the boundary of $\mathcal{CH}(Q)$, the proof is completed. If $y_{\mathcal{CH}(Q)}$ is an interior point of $\mathcal{CH}(Q)$, construct the line

$$L = \left\{ y_{\mathcal{CH}(Q)} + (y_S - y_{\mathcal{CH}(Q)})t \mid 0 \le t \le 1, t \in \mathbb{R} \right\} \subseteq \mathbb{R}^n \tag{5}$$

between the two points. Since $y_{\mathcal{CH}(Q)}$ is an interior point of a closed set, take $\delta \in \mathbb{R}_{>0}$ small enough so that $\hat{y} = y_{\mathcal{CH}(Q)} + (y_S - y_{\mathcal{CH}(Q)})\delta \in \mathcal{CH}(Q)$. This implies that

$$f((y_S, \hat{y})) = (1 - \delta) f(y) < f(y). \tag{6}$$

Note that $(y_S, \hat{y}) \in V$ hence this contradicts the fact that $y = \arg\min_{x \in V} f(x)$. Hence the optimal point $y_{\mathcal{CH}(Q)}$ must lie on the boundary of $\mathcal{CH}(Q)$. □

Denote by $E(Q)$ the set of extreme points of $\mathcal{CH}(Q)$, where $E(Q) \subseteq Q$. A simple inequality (Thereom 2), based only on the extreme points of Q, will now be derived, to quickly determine whether a point in Q can be the point closest to S. Such a criterion will allow us to neglect subsets Q that are far away from S without computing the distance from these points to S. In order to derive a lower bound on how close $x \in \mathcal{CH}(Q)$ can be to S a few definitions are necessary. In spite of the following quantities being dependent on Q and S, where there is no risk of confusion, we suppress this in their definitions because Q and S are fixed throughout the article.

Definition 3. Define $u \in Q$ to be the point that is closest to S,

$$u = \arg\min_{x \in Q} d(x, S). \tag{7}$$

Definition 4. Let $v \in E(Q)$ be the extreme point closest to S,

$$v = \arg\min_{x \in E(Q)} d(x, S). \tag{8}$$

Definition 5. Let $w \in E(Q)$ be the extreme point that is closest to u in Definition 3,

$$w = \arg\min_{x \in E(Q)} d(x, u). \tag{9}$$

Note that $u \neq v$ can hold since u may not be an extreme point. It is also possible that $u = v = w$ in which the closest point is an extreme point. A situation where all points are distinct is illustrated to the left in Fig. 1.